

# Bi-directional Analysis for Certification of Safety-Critical Software

Robyn R. Lutz\* and Robert M. Woodhouse  
Jet Propulsion Laboratory  
California Institute of Technology  
Pasadena, CA 91109

July 13, 1998

## Abstract

For safety-critical systems, it is insufficient to certify the developer and the development process. Certification of the software product itself is also needed. SFMEA (Software Failure Modes and Effects Analysis) and SFTA (Software Fault Tree Analysis) are two engineering techniques that have been used successfully for a number of years and in a variety of safety-critical applications to verify software design compliance with robustness and fault-tolerance standards. This paper proposes the use of Bi-directional Analysis (BDA), an integrated extension of SFMEA and SFTA, as a core assessment technique by which safety-critical software can be certified. BDA can provide limited but essential assurances that the software design has been systematically examined and complies with requirements for software safety.

## 1 Introduction

Even as requirements for software certification proliferate, the best approach to software certification remains in dispute [43]. One suggested piece of the solution is to certify the software developer or analyst. Another approach to software certification is to certify the development process (e.g., to assure compliance with ISO 9000-3, CMM or SPICE standards). For safety-critical software, a third approach, certifying the software product itself, is an essential aspect of software certification.

All three of these certification approaches (certifying the developer, the process, and the product) share a drive towards standardization (of credentials, of development processes, of software verification methods). All three approaches involve the necessary internationalization of certification as software production increasingly ignores national boundaries. All three approaches also involve independent evaluation of compliance of the developer/process/product against some pre-existing standard or guideline. However, since the

---

\*First author's address is Dept. of Computer Science, Iowa State University, Ames, IA 50011.

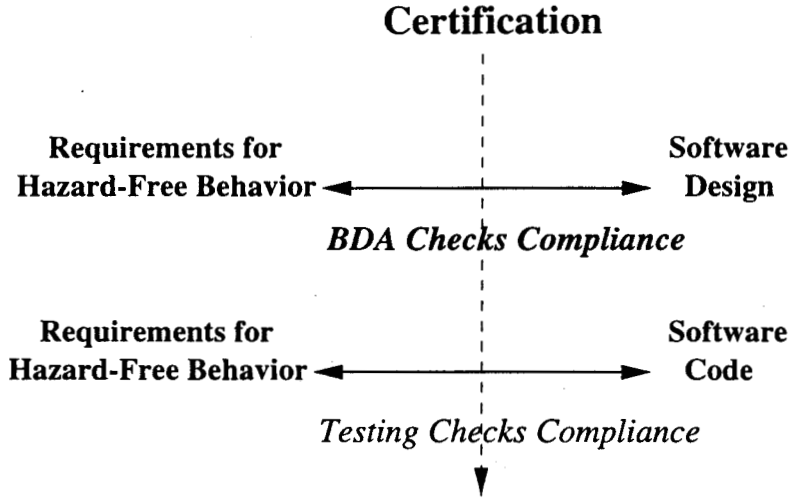


Figure 1: Role of Bi-directional Analysis (BDA) in Certification of Safety-Critical Software

## 2 Overview of BDA

BDA first checks the design to determine whether the effects of abnormal (e.g., out-of-range) input values and unexpected software events (e.g., unexpected termination) can contribute to unsafe system behavior. Following Leveson [15], software safety is defined to be freedom from undesired and unplanned events that results in a specified level of loss. Software safety analysis techniques determine how software can contribute to conditions that result in loss or failure. The forward direction of the BDA involves a forward analysis from abnormal inputs or events to non-compliant consequences, and has its roots in Failure Modes and Effects Analysis. The BDA then checks whether the non-compliant scenarios that have been identified are credible. This analysis either determines that the failure modes cannot occur given the design of this system or, if they can occur, that they are handled safely. This direction of the BDA involves a backward analysis from those abnormal scenarios with safety consequences to the collection of causes that might permit the identified scenario to happen. The second part of the BDA has its roots in Fault Tree Analysis.

For software there is no “seal of approval” that guarantees that software will behave safely. Instead, meaningful certification of safety-critical software is currently limited to a structured assessment, using well-documented techniques, that the software complies with certain specifications on its behavior [12]. For example, the ESPRIT2 project SCOPE (Software Certification Programme in Europe) pursued product certification by evaluating and assessing software compliance against requirements in a documented standard [40].

Fig. 1 shows the role of BDA (design certification) and testing (code certification) for safety-critical software. Code testing is the most important means of software certification. However, since testing is always partial and incomplete, assessment of design compliance with required behavior is also needed. In addition, design compliance can be assessed prior to testing, allowing needed changes to be made earlier in development.

Figure 2 shows an overview of the two components of a Bi-directional Analysis, the

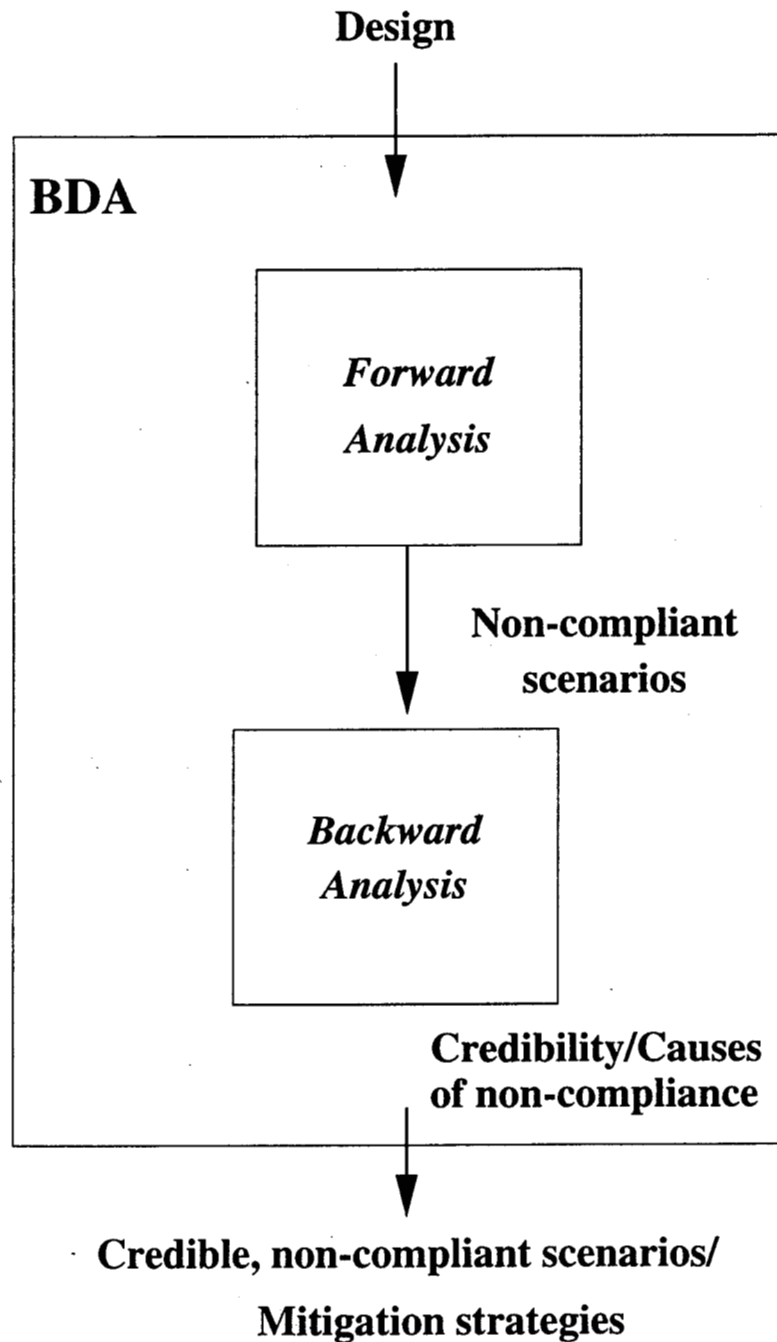


Figure 2: Bi-directional Analysis (BDA) Procedure

<i>Failure Mode</i>	<i>Failure Descrip</i>	<i>Local Effect</i>	<i>Sys Effect</i>	<i>Crit</i>	<i>Recomm</i>
Wrong timing	Sensor data outdated	Pump turned off	Temp limit exceeded	3	Test input timeliness

Table 1: Excerpt from BDA Forward Analysis

1, the Failure Mode type is “Wrong timing of data” the Failure Description column is, “Sensor input data received is incorrect,” the Local Effect column is, “Refrigerant pump is erroneously commanded off,” and the System Effect column describes the consequence for the system (“Temperature limit is exceeded”).

The next column indicates the “Criticality” of the failure mode’s effects. In the example, this column is “3” (of 5) indicating a threat to the subsystem but not the system. (A standard 5-part criticality measure reflects the severity of the failure’s effects, ranging from “no effect” to “catastrophic effect.”) The criticality column has safety implications in that high criticality ratings can be used to indicate the existence of hazards. On the other hand, non-critical effects, even if indicative of design errors, may not have safety implications. Depending on the standard against which compliance is being certified, it may be the case that only items with criticality ratings above some threshold will require further analysis.

The final column in the forward analysis table, “Recommendations” proposes corrective actions to eliminate the non-compliant scenario that has been identified. Often it makes sense to defer filling in this column until the second part of the BDA (the backward analysis to identify contributing causes of the failure mode) has been performed. In all but the simplest failure modes, it is often difficult to propose a meaningful corrective action until a better understanding of the circumstances surrounding the failure mode exist.

The second kind of table used for the forward analysis is an Events Table. An Events Table documents the effect of incorrect behavior or an incorrect event on the component and the system. The Events Table assists in the search for process failures, including the effects of software that fails to function correctly.

For each event (step in processing), each of the following four failure modes is analysed:

- Halt/Abnormal Termination (e.g., hung or deadlocked at this point)
- Omitted Event (e.g., event does not take place, but execution continues)
- Incorrect Logic (e.g., preconditions are inaccurate; event does not implement intent)
- Timing/Order (e.g., event occurs in wrong order; event occurs too early/late)

The Events Table documents the consequences of these failure modes for the events in the component under review and classifies the criticality of the effects. For example, the Failure Mode in one entry was “Timing/Order”, the Failure Description was “Instrument turned on too soon”, the Local Effect was “Insufficient power,” the System Effect was “Undervoltage occurs,” and the Criticality was rated “2” (since in this system, there was software to handle recovery from an undervoltage).

under review [23]. Automated tools to assist with portions of these analyses are currently being tested on requirements and design models. [25].

BDA is product-oriented rather than process-oriented in that it can “exercise and stress” the design of the software product [46]. It first checks the effect on the system of corrupted input or abnormal event execution without consideration of the source of error. Once a scenario is identified that leads to non-compliant output or behavior, BDA then traces backward in time to document the contributing causes for possible re-design or test.

## 4 Evaluation

Bi-directional analysis (BDA) has several advantages that recommend its use as a design certification technique to developers of safety-critical software.

- *Availability* The techniques BDA is based on are well-documented [10, 15, 18, 34, 44, 39], hence relatively easy to teach and apply. These features make the technique readily available.
- *Structure* The structured step-by-step procedure of BDA guides implementation, and the techniques involved are familiar to engineers worldwide.
- *Maintainability* The information developed during application of BDA analysis is broadly accessible since the format is readable, table-based, and web-accessible.
- *Safety Assessment* The role of BDA in the design certification process links clearly with requirements, since it assesses the adequacy of the software design in terms of satisfying the system safety requirements. BDA also provides forward links in the development process, since it prioritizes action items (i.e., recommendations for mitigating actions), prioritizes the hazards it uncovers (via the criticality measure), and provides a critical piece of the hazard analysis for the software design. BDA can also identify test cases in order to exercise each failure mode and confirm that the system reacts safely [29].
- *Incremental/evolutionary development* BDA analysis products fit into an incremental development process by being updatable for documentation. Initial work also suggests that for product line systems, a BDA of the product family can be largely reused in later instances of that family [17].
- *Independent certification* BDA can be used for design level certification against documented requirements specifications and has been widely used as a means for independent verification.
- *Systems focus* BDA is consistent with hardware certification practices (e.g., FMEA and FTA), thus encouraging a systems approach to safety.
- *Tools* BDA may also be amenable to automation. Automated tools exist for software forward and backward analyses [1, 7, 8, 26, 31], but their capabilities are limited. More powerful tools, incorporating forward and backward analyses, are currently being developed, e.g., by Safeware Engineering Corporation [33].

## Acknowledgments

The work described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process, or service by tradename, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

## References

- [1] Bell, D., L. Cox, S. Jackson and P. Schaefer (1992), "Using Causal Reasoning for Automated Failure Modes & Effects Analysis (FMEA)", *IEEE Proceedings of the Annual Reliability and Maintainability Symposium*, pp. 343-353.
- [2] Cha, S. S., N. G. Leveson, and T. J. Shimeall (1988), "Safety Verification in Murphy Using Fault Tree Analysis," in *Proc of the 10th International Conference on Software Engineering*, Apr, 1988, Singapore, pp. 377-386.
- [3] Chillarege, R., et al. (1992), "Orthogonal Defect Classification-A Concept for In-Process Measurements," *IEEE Transactions on Software Engineering*, 18, 11, 943-956.
- [4] Electronic Industries Association, (1983), "System Safety Analysis Techniques," Safety Engineering Bulletin No. 3-A, Engineering Department, Washington, D. C.
- [5] Electronic Industries Association, (1990) "System Safety Engineering in Software Development," Safety Engineering Bulletin No. 6-A, Engineering Department, Washington, D. C.
- [6] Fenelon, P. and J. A. McDermid (1993), "An Integrated Toolset for Software Safety Assessment," *Journal of Systems and Software*, July.
- [7] *FEAT (Failure Environment Analysis Tool)*, NASA Cosmic #MSC-21873.
- [8] *FIRM (Failure Identification and Risk Management Tool)*, Lockheed Engineering and Sciences Co., Cosmic.
- [9] Fragola, J. R. and J. F. Spahn (1973), "The Software Error Effects Analysis; A Qualitative Design Tool," in *Record, 1973 IEEE Symposium on Computer Software Reliability*, IEEE 73CH0741-9C, pp. 90-93.
- [10] Hall, F. M., R. A. Paul and W. E. Snow (1983), "Hardware/Software FMECA", *Proceedings of the Annual Reliability and Maintainability Symposium*, pp. 320- 327.
- [11] Herrman, D. S., (1995), "A methodology for evaluating, comparing, and selecting software safety and reliability standards," *Proceedings of the Tenth Annual Conference on Computer Assurance*, pp. 223-232.
- [12] *IEEE Standard Glossary of Software Engineering Terminology* (1990), IEEE Std 610.12-1990. New York: IEEE.

- [30] *Procedures for Performing a Failure Mode, Effects and Criticality Analysis* (1980), MIL-STD-1629A.
- [31] Pugh, D. R. and N. Snooke, (1996), "Dynamic Analysis of Qualitative Circuits for Failure Mode and Effects Analysis," *IEEE Proceedings of the Annual Reliability and Maintainability Symposium*, pp. 37-42.
- [32] Raheja, J. (1991) *Assurance Technologies: Principles and Practices*, McGraw-Hill.
- [33] Ratan, V., K. Partridge, J. Reese and N. Leveson, (1996), "Safety analysis tools for requirements specifications," *Proceedings of the Eleventh Annual Conference on Computer Assurance*, pp. 149-160.
- [34] Reifer, D. J. (1979), "Software Failure Modes and Effects Analysis," *IEEE Transactions on Reliability*, R-28, 3, 247-249.
- [35] Roland, H. E. and B. Moriarty, (1990), *System Safety Engineering and Management*. New York, New York: John Wiley and Sons.
- [36] RTCA/DO-178B (1992), *Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Inc.
- [37] Rushby, J. (1993) *Formal Methods and Digital Systems Validation for Airborne Systems*, SRI-CSL-93-07.
- [38] SAE (1996), *Aerospace Recommended Practice: Certification Considerations for Highly-Integrated or Complex Aircraft Systems*, ARP4754.
- [39] SAE (1996), *Aerospace Recommended Practice: Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment*, ARP4761.
- [40] SCOPE Project (1993), ESPRIT2, <http://www.cordis.lu/esprit/src/index.htm>
- [41] Selby, R. W. and V. R. Basili (1991), "Analyzing Error-Prone System Structure," *IEEE Transactions on Software Engineering*, 17, 2, 141-152.
- [42] Stephenson, J., (1991), *System Safety 2000: A practical guide for planning, managing and conducting system safety programs*. New York, New York: Van Nostrand Reinhold.
- [43] "Streamlining Software Aspects of Certification," <http://shemesh.larc.nasa.gov/ssac/>
- [44] System Safety Society (1993), *System Safety Analysis Handbook*.
- [45] Talbert, N. (1998), "The Cost of COTS: An Interview with John McDermid," *Computer*, 31, 6, June, 46-52.
- [46] Voas, J. (1998), "A Recipe for Certifying High Assurance Software," RST Corp., <http://www.rstcorp.com/paper-chrono.html>.
- [47] Wallace, D. R., L. M. Ippolito and D. R. Kuhn, (1992), "High Integrity Software Standards and Guidelines," Gaithersburg, MD: U.S. Department of Commerce, National Institute of Standards and Technology, NIST Special Publication 500-204, September.